Seminar Report: Paxy

Cegarra Dueñas, Gerard
Jabeen, Sana
Mosquera Dopico, Sergio

December 16, 2018

1 Introduction

This report intention consists on explaining Paxos algorithm, which is used to gain consensus in a distributed system. Paxos is a fault tolerant protocol for solving the consensus problem, where participants in a quorum need to agree on the chosen value.

Processes in Paxos algorithm can assume three different roles: proposers, acceptors and learners (the same process can assume any of these roles, but not at the same time). In this seminar we will just need to implement the proposer and the acceptor, because the learner is not needed to reach a consensus.

2 Work done

The work consisted on developing several experiments to ensure we have understand correctly the algorithm (and also the code). This is developed fully in Erlang, taking advantage of the material provided by the teacher. The code is sent along with this report and there is a general code (the original one) and several folders with changes in the code that represent the different experiments.

3 Experiments

3.1 Initial Sleep Time

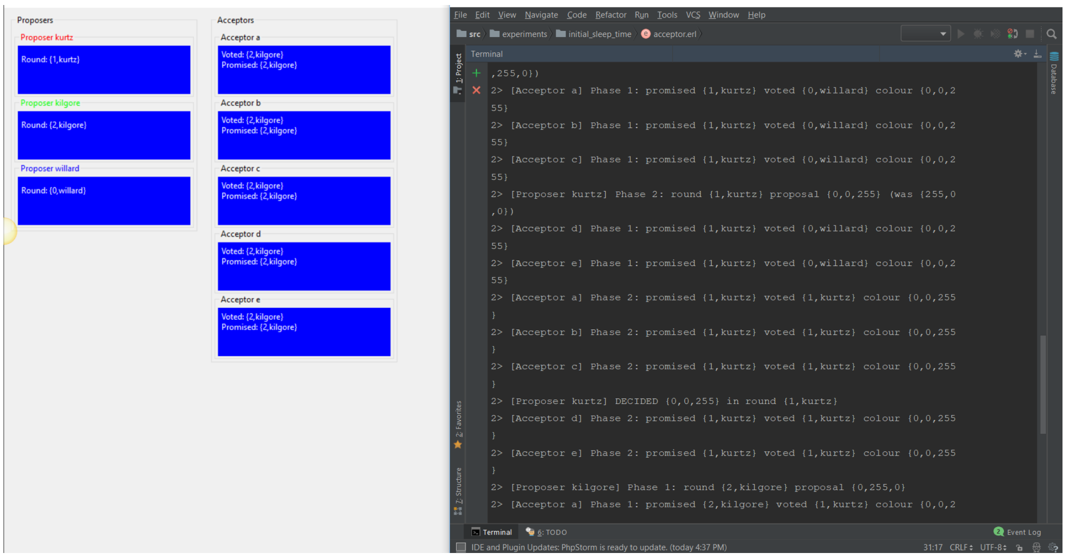
In this experiment we have changed the initial sleep time. For the first time the time for each proposer is same for this the command is:

```
paxy:start([200,200,200]).
```

Also by we have done another experiment by delaying the proposers the command are:

```
paxy:start([1000,2000,3000]).
paxy:start([2000,100,3000]).
```

By altering the delay time it could be seen that the proposer with same time are taking more time to reach consensus as compared if they are delayed. The result we got from this is in the following figures:



Proposers
Acceptors
File Edit View Navigate Code Refactor Run Tools VCS Window Help
src experiments initial_sleep_time acceptor.erl
Terminal
Terminal 6: TODO Event Log

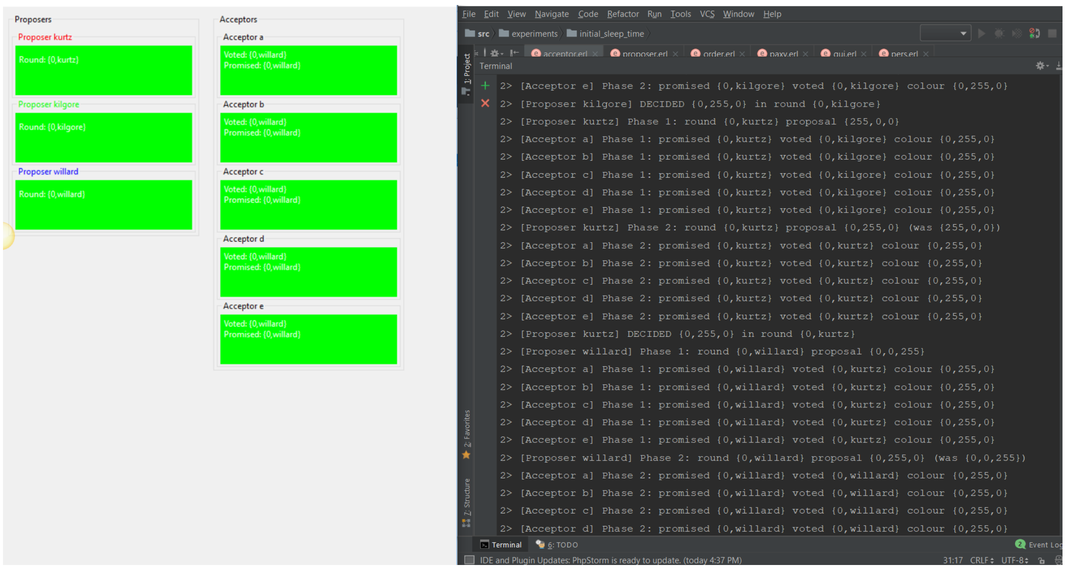
Proposer kurtz
Round: {1,kurtz}
Proposer kilgore
Round: {2,kilgore}
Proposer willard
Round: {0,willard}

Acceptor a
Voted: {2,kilgore}
Promised: {2,kilgore}
Acceptor b
Voted: {2,kilgore}
Promised: {2,kilgore}
Acceptor c
Voted: {2,kilgore}
Promised: {2,kilgore}
Acceptor d
Voted: {2,kilgore}
Promised: {2,kilgore}
Acceptor e
Voted: {2,kilgore}
Promised: {2,kilgore}

```
,255,0})
2> [Acceptor a] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,0,2
55}
2> [Acceptor b] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,0,2
55}
2> [Acceptor c] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,0,2
55}
2> [Proposer kurtz] Phase 2: round {1,kurtz} proposal {0,0,255} (was {255,0
,0})
2> [Acceptor d] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,0,2
55}
2> [Acceptor e] Phase 1: promised {1,kurtz} voted {0,willard} colour {0,0,2
55}
2> [Acceptor a] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Acceptor b] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Acceptor c] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Proposer kurtz] DECIDED {0,0,255} in round {1,kurtz}
2> [Acceptor d] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Acceptor e] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Proposer kilgore] Phase 1: round {2,kilgore} proposal {0,255,0}
2> [Acceptor a] Phase 1: promised {2,kilgore} voted {1,kurtz} colour {0,0,2
```

IDE and Plugin Updates: PhpStorm is ready to update. (today 4:37 PM) 31:17 CRLF UTF-8

Figure 1: Initial sleep Time with same proposers value



Proposers
Acceptors
File Edit View Navigate Code Refactor Run Tools VCS Window Help
src experiments initial_sleep_time
acceptor.erl proposer.erl order.erl paxy.erl gui.erl pers.erl
Terminal
Terminal 6: TODO Event Log

Proposer kurtz
Round: {0,kurtz}
Proposer kilgore
Round: {0,kilgore}
Proposer willard
Round: {0,willard}

Acceptor a
Voted: {0,willard}
Promised: {0,willard}
Acceptor b
Voted: {0,willard}
Promised: {0,willard}
Acceptor c
Voted: {0,willard}
Promised: {0,willard}
Acceptor d
Voted: {0,willard}
Promised: {0,willard}
Acceptor e
Voted: {0,willard}
Promised: {0,willard}

```
2> [Acceptor e] Phase 2: promised {0,kilgore} voted {0,kilgore} colour {0,255,0}
2> [Proposer kilgore] DECIDED {0,255,0} in round {0,kilgore}
2> [Proposer kurtz] Phase 1: round {0,kurtz} proposal {255,0,0}
2> [Acceptor a] Phase 1: promised {0,kurtz} voted {0,kilgore} colour {0,255,0}
2> [Acceptor b] Phase 1: promised {0,kurtz} voted {0,kilgore} colour {0,255,0}
2> [Acceptor c] Phase 1: promised {0,kurtz} voted {0,kilgore} colour {0,255,0}
2> [Acceptor d] Phase 1: promised {0,kurtz} voted {0,kilgore} colour {0,255,0}
2> [Acceptor e] Phase 1: promised {0,kurtz} voted {0,kilgore} colour {0,255,0}
2> [Proposer kurtz] Phase 2: round {0,kurtz} proposal {0,255,0} (was {255,0,0})
2> [Acceptor a] Phase 2: promised {0,kurtz} voted {0,kurtz} colour {0,255,0}
2> [Acceptor b] Phase 2: promised {0,kurtz} voted {0,kurtz} colour {0,255,0}
2> [Acceptor c] Phase 2: promised {0,kurtz} voted {0,kurtz} colour {0,255,0}
2> [Acceptor d] Phase 2: promised {0,kurtz} voted {0,kurtz} colour {0,255,0}
2> [Acceptor e] Phase 2: promised {0,kurtz} voted {0,kurtz} colour {0,255,0}
2> [Proposer kurtz] DECIDED {0,255,0} in round {0,kurtz}
2> [Proposer willard] Phase 1: round {0,willard} proposal {0,0,255}
2> [Acceptor a] Phase 1: promised {0,willard} voted {0,kurtz} colour {0,255,0}
2> [Acceptor b] Phase 1: promised {0,willard} voted {0,kurtz} colour {0,255,0}
2> [Acceptor c] Phase 1: promised {0,willard} voted {0,kurtz} colour {0,255,0}
2> [Acceptor d] Phase 1: promised {0,willard} voted {0,kurtz} colour {0,255,0}
2> [Acceptor e] Phase 1: promised {0,willard} voted {0,kurtz} colour {0,255,0}
2> [Proposer willard] Phase 2: round {0,willard} proposal {0,255,0} (was {0,0,255})
2> [Acceptor a] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
2> [Acceptor b] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
2> [Acceptor c] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
2> [Acceptor d] Phase 2: promised {0,willard} voted {0,willard} colour {0,255,0}
```

IDE and Plugin Updates: PhpStorm is ready to update. (today 4:37 PM) 31:17 CRLF UTF-8

Figure 2: Initial sleep Time with Different proposers value

3.2 Avoid sending sorry messages

In this experiment, we want to check if sorry messages may change the result or not. Initially we suppose they won't change the result because they are used just to inform the proposer about the situation (they can be omitted).

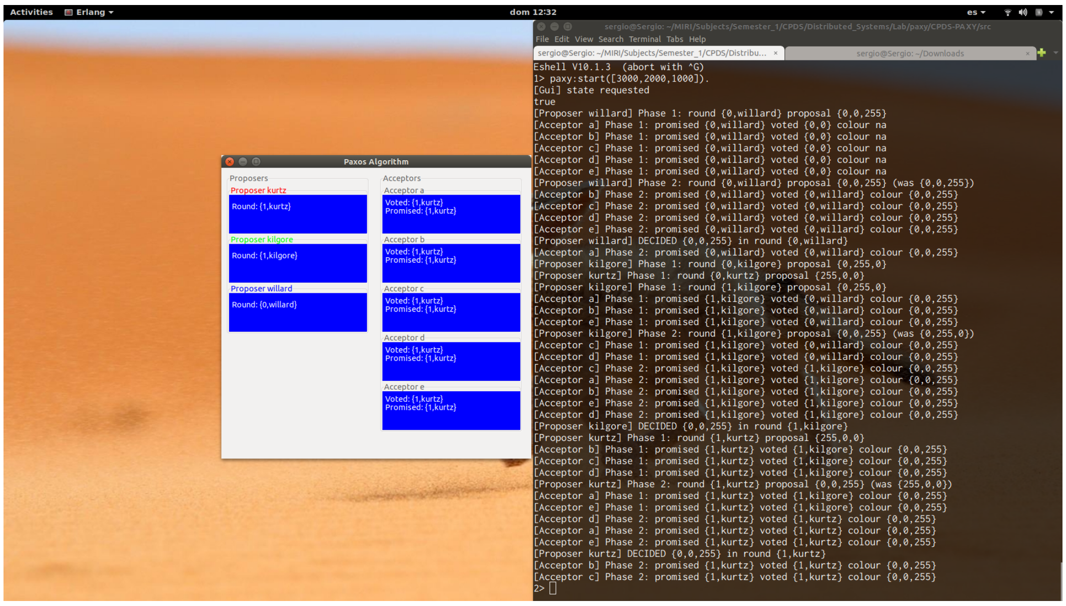
3.2.1 Steps to execute the experiment

For this experiment, we will run the following commands to test who wins:

```
paxy:start([1000,3000,2000]).
paxy:start([3000,2000,1000]).
paxy:start([2000,1000,3000]).
```

We will compare the two outputs, i.e. the number of rounds and the winner (name and color), for the different situations (with sorry message, and without sorry message) but with the same input. The code without the sorry messages will be contained on *src/experiments/no_sorry_msg*.

The results of the execution will be something like the following image:



```
Eshell V10.1.3  (abort with ^G)
1> paxy:start([3000,2000,1000]).
[Gui] state requested
true
[Proposer willard] Phase 1: round {0,willard} proposal {0,0,255}
[Acceptor a] Phase 1: promised {0,willard} voted {0,0} colour na
[Acceptor b] Phase 1: promised {0,willard} voted {0,0} colour na
[Acceptor c] Phase 1: promised {0,willard} voted {0,0} colour na
[Acceptor d] Phase 1: promised {0,willard} voted {0,0} colour na
[Acceptor e] Phase 1: promised {0,willard} voted {0,0} colour na
[Proposer willard] Phase 2: round {0,willard} proposal {0,0,255} (was {0,0,255})
[Acceptor b] Phase 2: promised {0,willard} voted {0,willard} colour {0,0,255}
[Acceptor c] Phase 2: promised {0,willard} voted {0,willard} colour {0,0,255}
[Acceptor d] Phase 2: promised {0,willard} voted {0,willard} colour {0,0,255}
[Acceptor e] Phase 2: promised {0,willard} voted {0,willard} colour {0,0,255}
[Proposer willard] DECIDED {0,0,255} in round {0,willard}
[Acceptor a] Phase 2: promised {0,willard} voted {0,willard} colour {0,0,255}
[Proposer kilgore] Phase 1: round {0,kilgore} proposal {0,255,0}
[Proposer kurtz] Phase 1: round {0,kurtz} proposal {255,0,0}
[Proposer kilgore] Phase 1: round {1,kilgore} proposal {0,255,0}
[Acceptor a] Phase 1: promised {1,kilgore} voted {0,willard} colour {0,0,255}
[Acceptor b] Phase 1: promised {1,kilgore} voted {0,willard} colour {0,0,255}
[Acceptor e] Phase 1: promised {1,kilgore} voted {0,willard} colour {0,0,255}
[Proposer kilgore] Phase 2: round {1,kilgore} proposal {0,0,255} (was {0,255,0})
[Acceptor c] Phase 1: promised {1,kilgore} voted {0,willard} colour {0,0,255}
[Acceptor d] Phase 1: promised {1,kilgore} voted {0,willard} colour {0,0,255}
[Acceptor c] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {0,0,255}
[Acceptor a] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {0,0,255}
[Acceptor b] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {0,0,255}
[Acceptor e] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {0,0,255}
[Acceptor d] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {0,0,255}
[Proposer kilgore] DECIDED {0,0,255} in round {1,kilgore}
[Proposer kurtz] Phase 1: round {1,kurtz} proposal {255,0,0}
[Acceptor b] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Acceptor c] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Acceptor d] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Proposer kurtz] Phase 2: round {1,kurtz} proposal {0,0,255} (was {255,0,0})
[Acceptor a] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Acceptor e] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,255}
[Acceptor d] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
[Acceptor a] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
[Acceptor e] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
[Proposer kurtz] DECIDED {0,0,255} in round {1,kurtz}
[Acceptor b] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
[Acceptor c] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255}
2> 
```

Figure 3: Sample execution

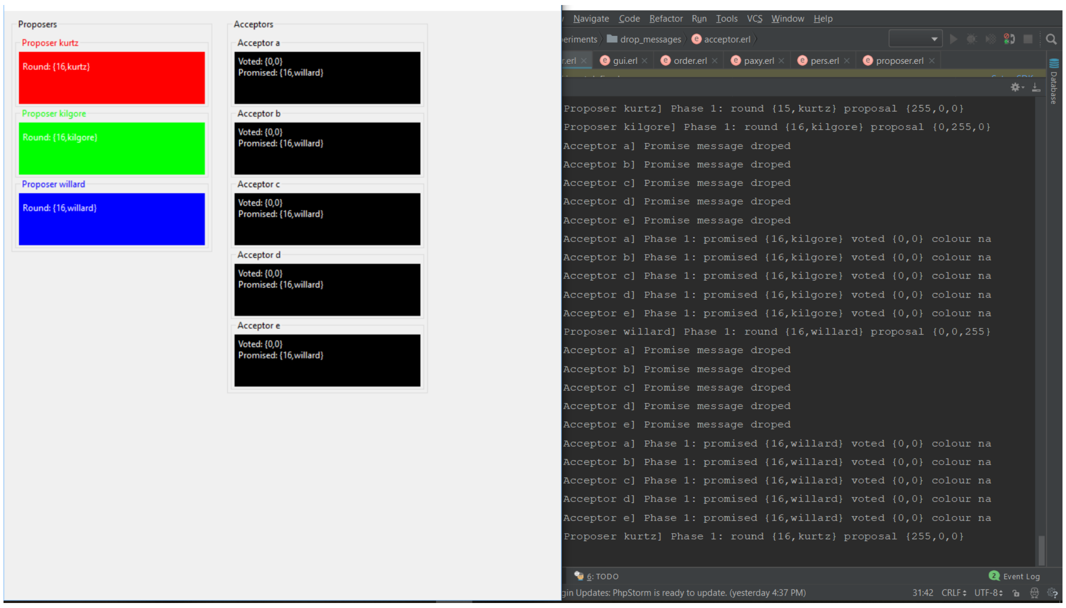
3.2.2 Results

In this case our predictions are fulfilled and we can ensure that we obtain the same results using a version with sorry message and without sorry message. For that reason we will not include two tables to show the same results:

	# OF ROUNDS	WINNER	COLOR
[1000,3000,2000]	1	KILGORE	RED
[3000,2000,1000]	1	KURTZ	BLUE
[2000,1000,3000]	0	WILLARD	GREEN

3.3 Randomly Dropping Messages in the Acceptor

In this experiment the promise and vote messages has been dropped. The result of which could be seen in the following figure. The algorithm will not reach to any consensus if both the vote and promise messages are dropped.


Proposers
Proposer kurtz
Proposer kilgore
Proposer willard
Acceptors
Acceptor a
Acceptor b
Acceptor c
Acceptor d
Acceptor e
Navigate Code Refactor Run Tools VCS Window Help
eriments drop_messages acceptor.erl
r.erl gui.erl order.erl paxy.erl pers.erl proposer.erl
Database
6: TODO
Event Log
gin Updates: PhpStorm is ready to update. (yesterday 4:37 PM)
31:42 CRLF UTF-8

Round: {16,kurtz}

Round: {16,kilgore}

Round: {16,willard}

Voted: {0,0}
Promised: {16,willard}

Voted: {0,0}
Promised: {16,willard}

Voted: {0,0}
Promised: {16,willard}

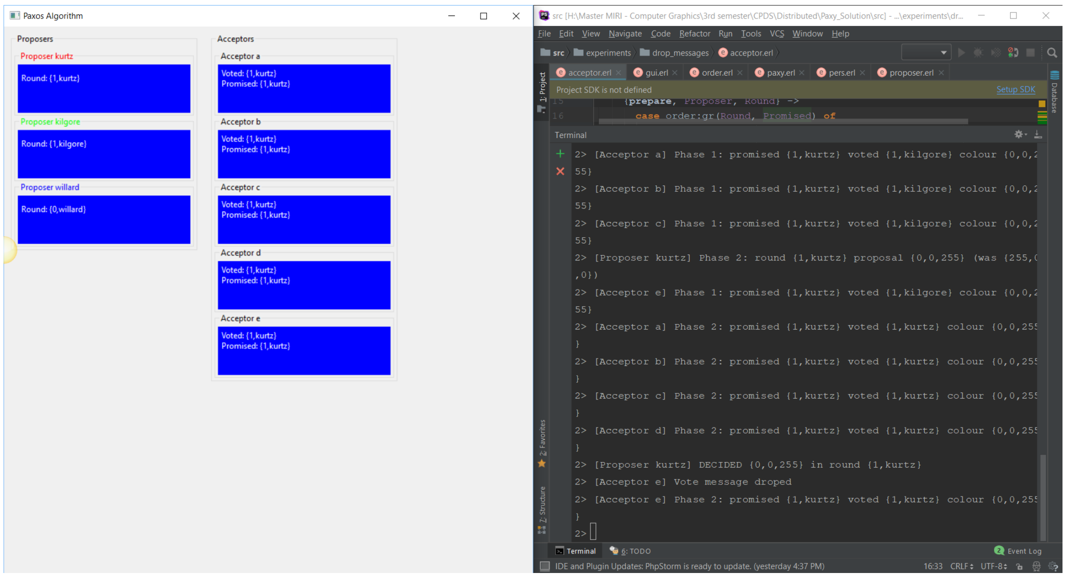
Voted: {0,0}
Promised: {16,willard}

Voted: {0,0}
Promised: {16,willard}

```
Proposer kurtz] Phase 1: round {15,kurtz} proposal {255,0,0}
Proposer kilgore] Phase 1: round {16,kilgore} proposal {0,255,0}
Acceptor a] Promise message droped
Acceptor b] Promise message droped
Acceptor c] Promise message droped
Acceptor d] Promise message droped
Acceptor e] Promise message droped
Acceptor a] Phase 1: promised {16,kilgore} voted {0,0} colour na
Acceptor b] Phase 1: promised {16,kilgore} voted {0,0} colour na
Acceptor c] Phase 1: promised {16,kilgore} voted {0,0} colour na
Acceptor d] Phase 1: promised {16,kilgore} voted {0,0} colour na
Acceptor e] Phase 1: promised {16,kilgore} voted {0,0} colour na
Proposer willard] Phase 1: round {16,willard} proposal {0,0,255}
Acceptor a] Promise message droped
Acceptor b] Promise message droped
Acceptor c] Promise message droped
Acceptor d] Promise message droped
Acceptor e] Promise message droped
Acceptor a] Phase 1: promised {16,willard} voted {0,0} colour na
Acceptor b] Phase 1: promised {16,willard} voted {0,0} colour na
Acceptor c] Phase 1: promised {16,willard} voted {0,0} colour na
Acceptor d] Phase 1: promised {16,willard} voted {0,0} colour na
Acceptor e] Phase 1: promised {16,willard} voted {0,0} colour na
Proposer kurtz] Phase 1: round {16,kurtz} proposal {255,0,0}
```

Figure 4: Voted and Promised messages dropped

By increasing the number of drop the algorithm reach to consensus immediately with round 0 as shown in the following fig.



```
    {prepare, Proposer, Round} ->
      case order:gr(Round, Promised) of
```

```
2> [Acceptor a] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,2
55}
2> [Acceptor b] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,2
55}
2> [Acceptor c] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,2
55}
2> [Proposer kurtz] Phase 2: round {1,kurtz} proposal {0,0,255} (was {255,0
,0})
2> [Acceptor e] Phase 1: promised {1,kurtz} voted {1,kilgore} colour {0,0,2
55}
2> [Acceptor a] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Acceptor b] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Acceptor c] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Acceptor d] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> [Proposer kurtz] DECIDED {0,0,255} in round {1,kurtz}
2> [Acceptor e] Vote message droped
2> [Acceptor e] Phase 2: promised {1,kurtz} voted {1,kurtz} colour {0,0,255
}
2> 
```

IDE and Plugin Updates: PhpStorm is ready to update. (yesterday 4:37 PM)

Figure 5: By increasing drop: Voted and Promised messages dropped

3.4 Split the PAXY Module

For this experiment we are required to change some aspects of the main code to allow the proposers and the acceptors to execute in two different system and to be able to communicate with each other as usual.

To achieve the final result, it was necessary to change three modules:

1. *gui.erl*. In this case we need to divide the implementation, creating two different windows, one for the proposers and the other one for the acceptors.

2. *paxy.erl*. In the same way we did in the *gui* module, we have to split the responsibilities. We need to initialize separately the acceptors and the proposers. Once both are created, will be the acceptors the ones in charge of establishing the connection with the proposers by means of a registry message.

3. *proposer.erl*. Finally, we need to change almost all the methods to include a reference to the program identifier. This extra parameter on each method is required to allow the message passing between proposers and acceptors

All the code is available in the folder *src/experiments/split_paxy*.

3.4.1 Steps to execute the experiment

To execute this code we need to open two terminals (or command lines) and execute the following commands, one on each terminal:

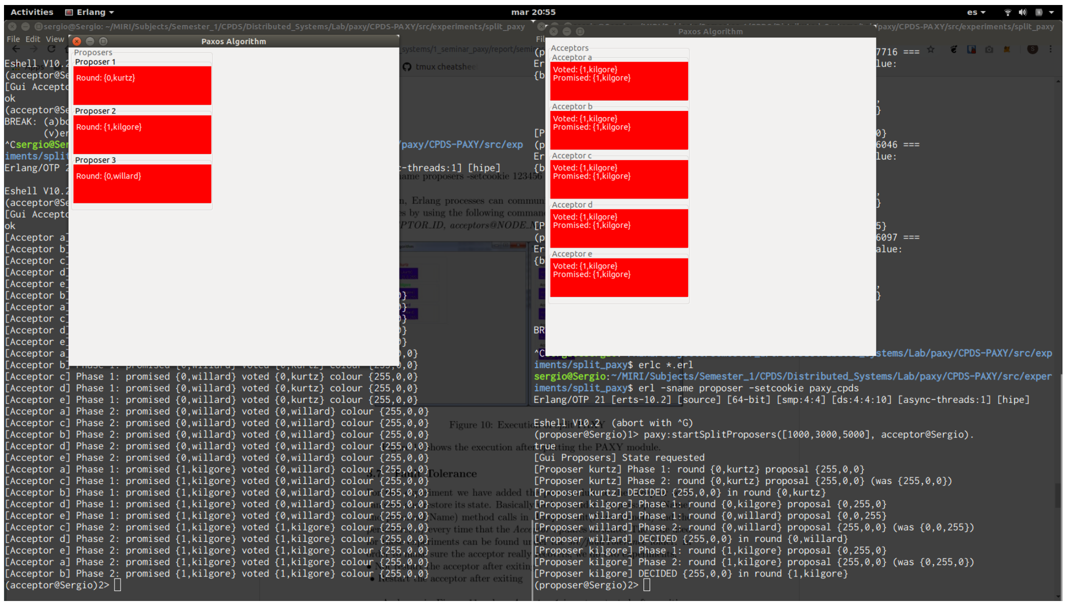
```
erl −sname paxy−acc −setcookie paxy_cpds
erl −sname paxy−pro −setcookie paxy_cpds
```

When we specify the *−sname*, is because we want to give a name to the program we are executing, and with *−setcookie* we specify a password or secret that allows us to isolate in a given group those programs that we want to act together.

Once we have these two programs running in different terminals, we run the following commands in the proposers and acceptors terminals, respectively:

```
paxy:startSplitAcceptors(paxy−acc@User).
paxy:startSplitProposers([t_a, t_b, t_c], paxy−pro@User).
```

In this case we can try different situations with sleep values (t_a, t_b, t_c) for a, b and c as we did in section 3.2 to check the execution of the program.



```
[Acceptor c] Phase 1: promised {0,willard} voted {0,kurtz} colour {255,0,0}
[Acceptor d] Phase 1: promised {0,willard} voted {0,kurtz} colour {255,0,0}
[Acceptor e] Phase 1: promised {0,willard} voted {0,kurtz} colour {255,0,0}
[Acceptor a] Phase 2: promised {0,willard} voted {0,willard} colour {255,0,0}
[Acceptor c] Phase 2: promised {0,willard} voted {0,willard} colour {255,0,0}
[Acceptor b] Phase 2: promised {0,willard} voted {0,willard} colour {255,0,0}
[Acceptor d] Phase 2: promised {0,willard} voted {0,willard} colour {255,0,0}
[Acceptor e] Phase 2: promised {0,willard} voted {0,willard} colour {255,0,0}
[Acceptor a] Phase 1: promised {1,kilgore} voted {0,willard} colour {255,0,0}
[Acceptor c] Phase 1: promised {1,kilgore} voted {0,willard} colour {255,0,0}
[Acceptor b] Phase 1: promised {1,kilgore} voted {0,willard} colour {255,0,0}
[Acceptor d] Phase 1: promised {1,kilgore} voted {0,willard} colour {255,0,0}
[Acceptor e] Phase 1: promised {1,kilgore} voted {0,willard} colour {255,0,0}
[Acceptor c] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {255,0,0}
[Acceptor d] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {255,0,0}
[Acceptor e] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {255,0,0}
[Acceptor a] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {255,0,0}
[Acceptor b] Phase 2: promised {1,kilgore} voted {1,kilgore} colour {255,0,0}
(acceptor@Sergio)2>
```

```
iments/split_paxy$ erlc *.erl
sergio@Sergio:~/MIRI/Subjects/Semester_1/CPDS/Distributed_Systems/Lab/paxy/CPDS-PAXY/src/exper
iments/split_paxy$ erl -sname proposer -setcookie paxy_cpds
Erlang/OTP 21 [erts-10.2] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [hipe]

Eshell V10.2  (abort with ^G)
(proposer@Sergio)1> paxy:startSplitProposers([1000,3000,5000], acceptor@Sergio).
true
[Gui Proposers] State requested
[Proposer kurtz] Phase 1: round {0,kurtz} proposal {255,0,0}
[Proposer kurtz] Phase 2: round {0,kurtz} proposal {255,0,0} (was {255,0,0})
[Proposer kurtz] DECIDED {255,0,0} in round {0,kurtz}
[Proposer kilgore] Phase 1: round {0,kilgore} proposal {0,255,0}
[Proposer willard] Phase 1: round {0,willard} proposal {0,0,255}
[Proposer willard] Phase 2: round {0,willard} proposal {255,0,0} (was {0,0,255})
[Proposer willard] DECIDED {255,0,0} in round {0,willard}
[Proposer kilgore] Phase 1: round {1,kilgore} proposal {0,255,0}
[Proposer kilgore] Phase 2: round {1,kilgore} proposal {255,0,0} (was {0,255,0})
[Proposer kilgore] DECIDED {255,0,0} in round {1,kilgore}
(proposer@Sergio)2>
```

Figure 6: Sample execution

3.5 Adding delays in the acceptor

For this question we will use a set up of 5 proposers and three acceptors. We will test three scenarios: adding a delay after receive a prepare, after receive an accept and both.

For each one of these three scenarios we will run 10 executions of the modified code. After each execution we will average the number of rounds that took to reach a consensus among the three processors.

Execution	**Delay after prepare**	**Delay after accept**	**Both**
1	1	2/3	2/3
2	1	2/3	2/3
3	2/3	2/3	2/3
4	1	2/3	2/3
5	2/3	2/3	2/3
6	2/3	2/3	2/3
7	2/3	2/3	2/3
8	2/3	2/3	2/3
9	1	2/3	2/3
10	2/3	2/3	2/3

So the average number of rounds to get consensus in these three scenarios is: 0.8, 0.67 and 0.67. If we apply this process to the original scenario where no delay was applied at all, we get a final average of 0.83.

3.5.1 Does the algorithm still terminate?

Yes, it terminated in every single execution.

3.5.2 Does it require more rounds?

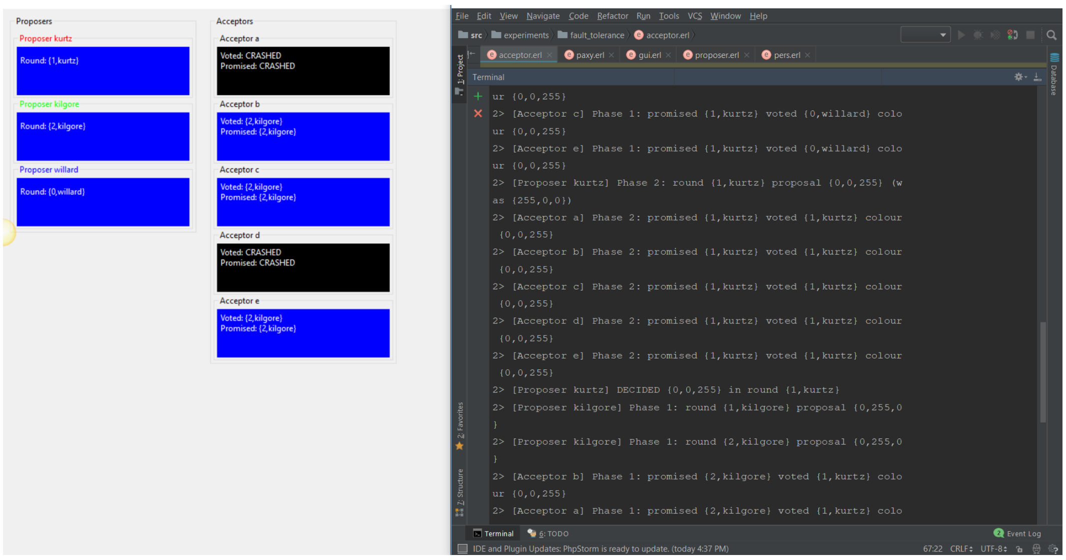
After running the experiment, we can come up with the conclusion that with the kind of delays that we implemented, the processors need less rounds to reach a consensus.

3.5.3 How does the impact of adding delays depend on the value of the timeout at the proposer?

If the delay in the acceptor is greater than the proposer's timeout, the communication proposer $\rightarrow$ acceptor is not possible. The proposer will timeout before the acceptor starts to send the response.

3.6 Fault Tolerance

In this experiment we have test if the acceptor recover or not. For that we have use pers module in which pers:open(), pers:read() and pers:store() methods are called in the Acceptor to store the updated state of the acceptor. The following figures shows the fault tolerance working well because if the acceptor starts and crashed after restarting it's working normally.


Proposers

Proposer kurtz
Round: {1,kurtz}

Proposer kilgore
Round: {2,kilgore}

Proposer willard
Round: {0,willard}

Acceptors

Acceptor a
Voted: CRASHED
Promised: CRASHED

Acceptor b
Voted: {2,kilgore}
Promised: {2,kilgore}

Acceptor c
Voted: {2,kilgore}
Promised: {2,kilgore}

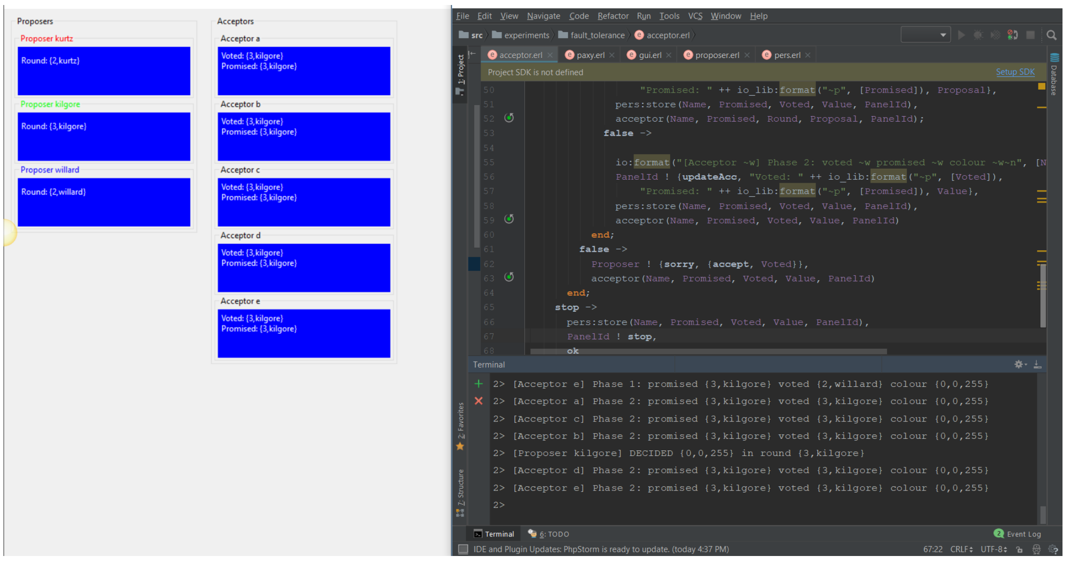
Acceptor d
Voted: CRASHED
Promised: CRASHED

Acceptor e
Voted: {2,kilgore}
Promised: {2,kilgore}

```
ur {0,0,255}
2> [Acceptor c] Phase 1: promised {1,kurtz} voted {0,willard} colo
ur {0,0,255}
2> [Acceptor e] Phase 1: promised {1,kurtz} voted {0,willard} colo
ur {0,0,255}
2> [Proposer kurtz] Phase 2: round {1,kurtz} proposal {0,0,255} (w
as {255,0,0})
2> [Acceptor a] Phase 2: promised {1,kurtz} voted {1,kurtz} colour
 {0,0,255}
2> [Acceptor b] Phase 2: promised {1,kurtz} voted {1,kurtz} colour
 {0,0,255}
2> [Acceptor c] Phase 2: promised {1,kurtz} voted {1,kurtz} colour
 {0,0,255}
2> [Acceptor d] Phase 2: promised {1,kurtz} voted {1,kurtz} colour
 {0,0,255}
2> [Acceptor e] Phase 2: promised {1,kurtz} voted {1,kurtz} colour
 {0,0,255}
2> [Proposer kurtz] DECIDED {0,0,255} in round {1,kurtz}
2> [Proposer kilgore] Phase 1: round {1,kilgore} proposal {0,255,0
}
2> [Proposer kilgore] Phase 1: round {2,kilgore} proposal {0,255,0
}
2> [Acceptor b] Phase 1: promised {2,kilgore} voted {1,kurtz} colo
ur {0,0,255}
2> [Acceptor a] Phase 1: promised {2,kilgore} voted {1,kurtz} colo
```

IDE and Plugin Updates: PhpStorm is ready to update. (today 4:37 PM)

Figure 7: Starts and crashed paxy:crash(a).


Proposers

Proposer kurtz
Round: {2,kurtz}

Proposer kilgore
Round: {3,kilgore}

Proposer willard
Round: {2,willard}

Acceptors

Acceptor a
Voted: {3,kilgore}
Promised: {3,kilgore}

Acceptor b
Voted: {3,kilgore}
Promised: {3,kilgore}

Acceptor c
Voted: {3,kilgore}
Promised: {3,kilgore}

Acceptor d
Voted: {3,kilgore}
Promised: {3,kilgore}

Acceptor e
Voted: {3,kilgore}
Promised: {3,kilgore}

Project SDK is not defined

```
                    "Promised: " ++ io_lib:format("~p", [Promised]), Proposal},
                pers:store(Name, Promised, Voted, Value, PanelId),
                acceptor(Name, Promised, Round, Proposal, PanelId);
              false ->

                io:format("[Acceptor ~w] Phase 2: voted ~w promised ~w colour ~w~n", [N
                PanelId ! {updateAcc, "Voted: " ++ io_lib:format("~p", [Voted]),
                    "Promised: " ++ io_lib:format("~p", [Promised]), Value},
                pers:store(Name, Promised, Voted, Value, PanelId),
                acceptor(Name, Promised, Voted, Value, PanelId)
            end;
          false ->
            Proposer ! {sorry, {accept, Voted}},
            acceptor(Name, Promised, Voted, Value, PanelId)
        end;
    stop ->
        pers:store(Name, Promised, Voted, Value, PanelId),
        PanelId ! stop,
        ok
```

```
2> [Acceptor e] Phase 1: promised {3,kilgore} voted {2,willard} colour {0,0,255}
2> [Acceptor a] Phase 2: promised {3,kilgore} voted {3,kilgore} colour {0,0,255}
2> [Acceptor c] Phase 2: promised {3,kilgore} voted {3,kilgore} colour {0,0,255}
2> [Acceptor b] Phase 2: promised {3,kilgore} voted {3,kilgore} colour {0,0,255}
2> [Proposer kilgore] DECIDED {0,0,255} in round {3,kilgore}
2> [Acceptor d] Phase 2: promised {3,kilgore} voted {3,kilgore} colour {0,0,255}
2> [Acceptor e] Phase 2: promised {3,kilgore} voted {3,kilgore} colour {0,0,255}
2>
```

IDE and Plugin Updates: PhpStorm is ready to update. (today 4:37 PM)

Figure 8: Recover crashed acceptors after restart

3.7 Increasing the number of acceptors and proposers

For this experiment we will be setting up different scenarios playing with the number of acceptors and proposers. As we did in the last experiment, for each scenario we will run 5 executions, and in each execution, calculate the average number of rounds it takes to reach a consensus among all proposers. Finally, for each scenario, we average the results of the 5 executions.

In first place we will be increasing the number of acceptors, from 5 to 15.

Setup [Proposers, Acceptors]	[3, 5]	[3, 10]	[3, 15]
Rounds	0.83	0.86	0.933

In Second place we increment the number of proposers, from 3 to 9.

Setup [Proposers, Acceptors]	[3, 5]	[6, 5]	[9, 5]
Rounds	0.83	1.36	2.16

3.8 What is the impact of increasing the number of either acceptors or proposers?

From the results we can conclude that increasing the number of acceptors it doesn't have any notable impact in the number of rounds needed to reach consensus, while increasing the proposers do. Actualy, The bigger is the number of proposers in the system, the bigger is the number of rounds needed.

3.9 Improvement based on sorry messages

To implement the improvement proposed in this section, we just need to count the number of sorry messages (relevant to the current round) that a proposer receives in the collect or vote process, and abort the ballot if that number overcomes the needed quorum.

Once we have implemented it, we discover that with this new optimization, the average number of rounds needed to get consensus remains the same, but the execution time of these rounds decreases notably. This is due to the proposers jump quicker to the next round.

We execute five times a set up of 9 proposers and 15 acceptors to calculate the average execution time. The averaged and rounded execution times are 10 for the non optimized version and 4 for the optimized one.

4 Open questions

1. **Avoid sending sorry messages by commenting the corresponding sentences in the acceptor. Q) Could you even come to an agreement when sorry messages are not sent?** *Yes, it is possible. As we stated in section 3.2, sorry messages are optional and are being used only to inform the proposers about the situation.*

2. **Try randomly dropping promise and/or vote messages in the acceptor. If you drop too many messages a quorum will**

of course never be found, but we could probably lose quite many. Q) What percentage of messages can we drop until consensus is not longer possible? In our experiment there are 16 rounds for which the messages are 16*10 that dropped with no consensus. You can drop messages using the following code, which will drop in average one in 10 messages. Try different drop ratios.

```
-define(drop, 1).
P = rand:uniform(10),
if P =< ?drop ->
io:format("message dropped~n");
true ->
    %send message
end.
```

This is explained in section 3.3

3. **Adapt the paxy module to create the proposers in a remote Erlang instance (named paxy-pro) and to ensure that they can connect correctly to the acceptors, which must be created in a different remote Erlang instance (named paxy-acc). Note that the acceptors have to use locally registered names. Remember how processes are created remotely, how names registered in remote nodes are referred, and how Erlang runtime should be started to run distributed programs.** *This is explained in section 3.4*

5 Personal opinion

It was a good chance to learn an important (and quite hard to learn at first) algorithm as Paxos. Encouraging the students to practice with it, by means of developing different experiments and interacting with the code is quite a good way to learn things that in some cases may be harder.

In our opinion we would keep the idea for the following courses, it makes it easier to understand the algorithm.